WaveVision-5 System Developers' Guide

Robbie Shergill
Ken Tateno
Rev. 0.91
July 17 2009

1.0 Overview

This document provides a comprehensive set of requirements and guidelines for engineers developing hardware or software within the WaveVision-5 system. The complete developers' guide is constituted of a set of documents. This document is the parent document and the following supporting documents are part and parcel of the complete developers' guide:

1. *SigPathData Interface specification.*
2. *WV5 HSP and FB Port Specification.*
3. *WV5 Core Naming Convention.*
4. *WV5 Core image map Spec.*

Those looking for an overview of the WaveVision system for general informational purposes should consult *"An Introduction to the WaveVision Signal Path Evaluation System"*.

An engineer wanting to design a DUT (device under test) board need only refer to Section 5 of this document. Section 7 describes some basic utilities that are necessary when developing hardware/software in the WV5 environment. In order to also customize the WaveVision-5 software to provide the appropriate device control to the user, one would need to refer to Section 8 as well. Controller board designers are guided to the SigPathData Interface Specification in Section 6.

In this document, the strict requirements use the terms "shall" and "shall not", while the guidelines use other terms such as "should".

With respect to software, this document applies to rev. 5.0.5.175 of the WV5 GUI installation and later.

2.0 References

a) WaveVision5 Software Users' Guide.

b) WaveVision-5.1 Capture Board Users' Guide.

c) Programmer's Guide to WV5_DLL API.

d) Programmer's Guide to WV5 C-Script.

3.0 Contents

4.0 Definitions

DUT Board:	A board that plugs into a standardized WaveVision-5.x digital controller board. It may be a device evaluation board or a signal-path reference board.
WV5 FB Port	The standard means to connect with National signal-path eval boards. Based on the FutureBus connector. The "port" spec defines the signals in addition to the mechanical and electrical specification.
WV5 HSP Port	The standard means to connect with higher-speed National signal-path eval boards that utilize serial data/clock interfaces. Based on the HMZd connector. The "port" spec defines the signals in addition to the mechanical and electrical specification.
SigPathData Interface	Standardized logical interface between the micro-controller and the rest of the hardware on the controller boards (primarily the FPGA - when present).
SPI-1.x	A "standardized" implementation of the SPI interface within National.
WV5 Core	Lower-level software and firmware residing below the application software and above the hardware boundary marked by the SigPathData interface. See Figure 6.

5.0 Designing the DUT Board

IMPORTANT NOTE: To guarantee compliance of a DUT board design with the WV5 controller card hardware, the schematic of the WV5 board should not be relied upon. This document and its associated documents form the definitive specification for compliance.

5.1 DUT Board Mechanicals

The mechanical requirements center around the positioning and the orientation of the connectors. The X-Y dimensions of the DUT board are not specified by this document.

There are 3 possible connector arrangements: a board with the FB (FutureBus) connector only; a board with the HMZd connector only; and a board with both connectors. The following three diagrams show the exact placement of the connectors from the mating edge of the DUT board that must be followed for each of these cases.

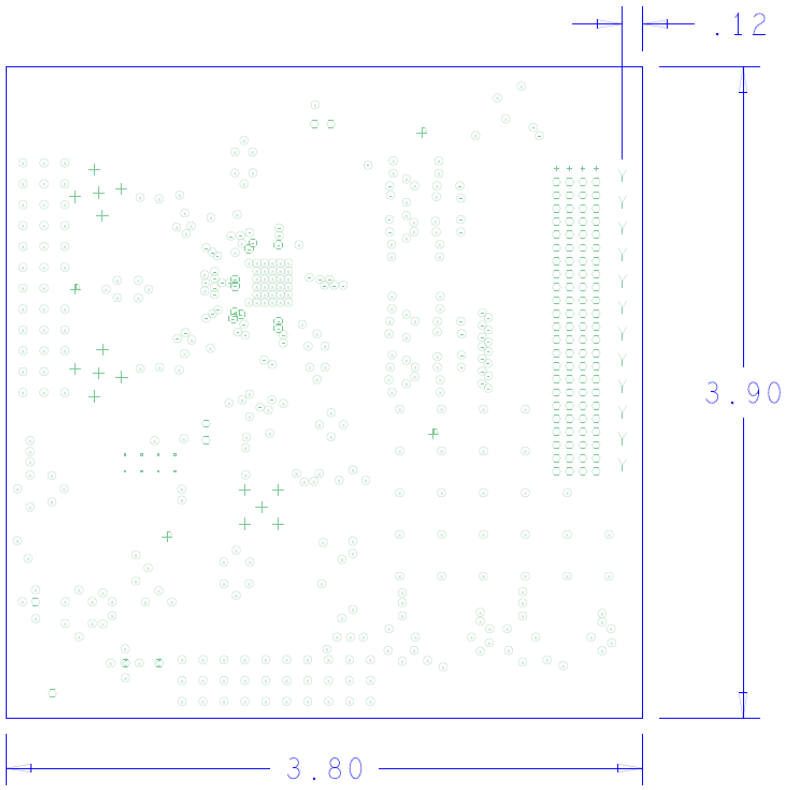


Figure 1: A board with the FB connector only (units of inches)
The 3.80 x 3.90 inch dimensions of the board are unimportant to the specification.

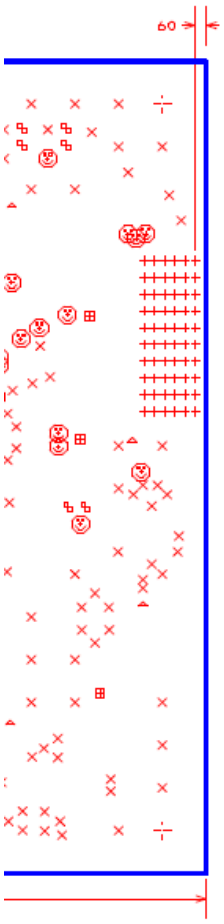


Figure 2: A board with the HMZd connector only (units of mils)

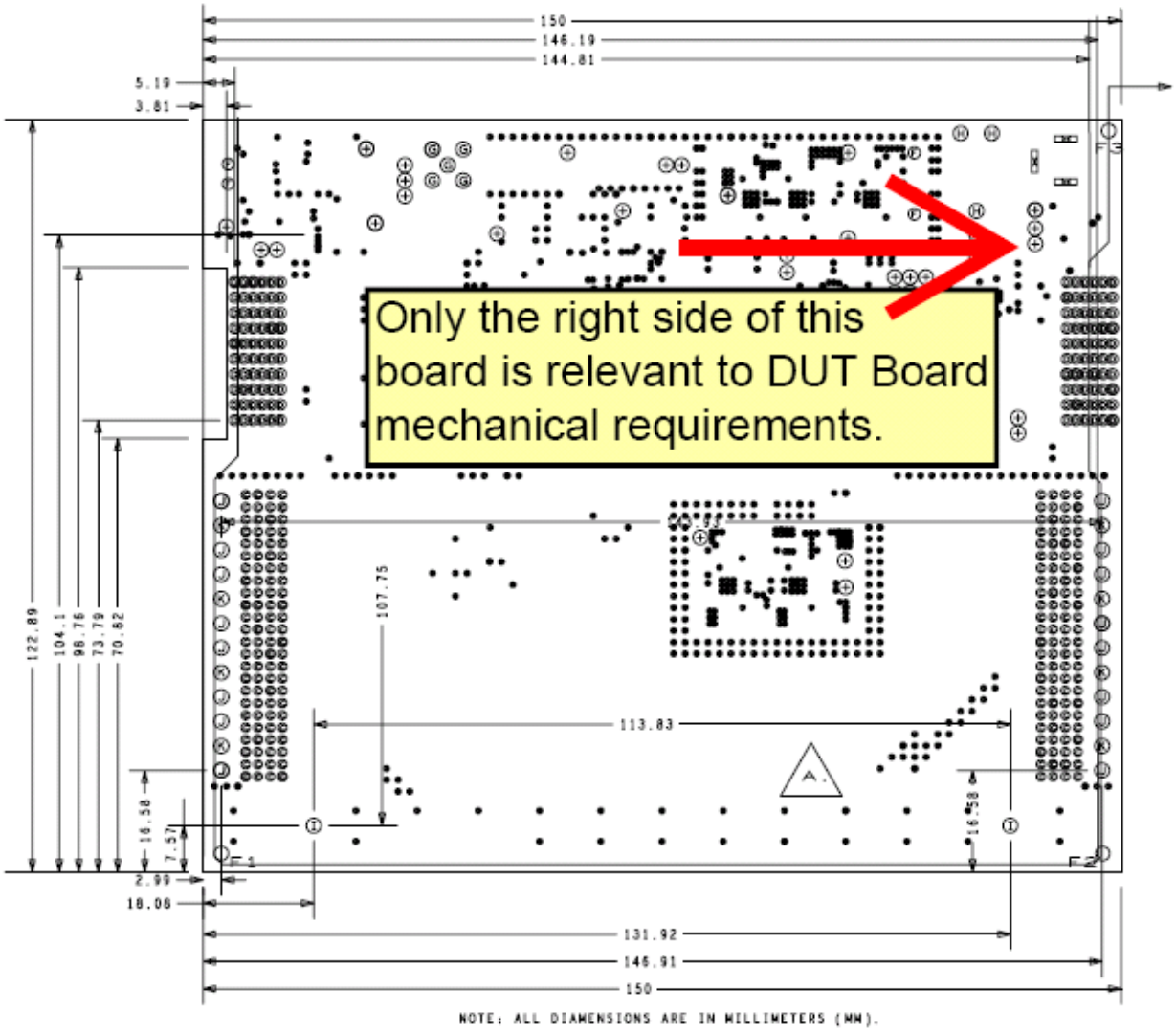


Figure 3: A board with both FB and HMZd connectors

Note that the HMZd connector's critical dimension is from the edge of the board to the center of the connector's footprint (=150-144.81 =5.19mm) while the critical dimension shown for the FB connector is from the board edge to the drill hole centers (= 150-146.91 = 3.09mm).

5.1.1 FB Connector Orientation

Though available in 24-pin sections, the FutureBus (FB) connector shall be used in the full 96-pin configuration and oriented on the DUT board as illustrated in the following diagram.

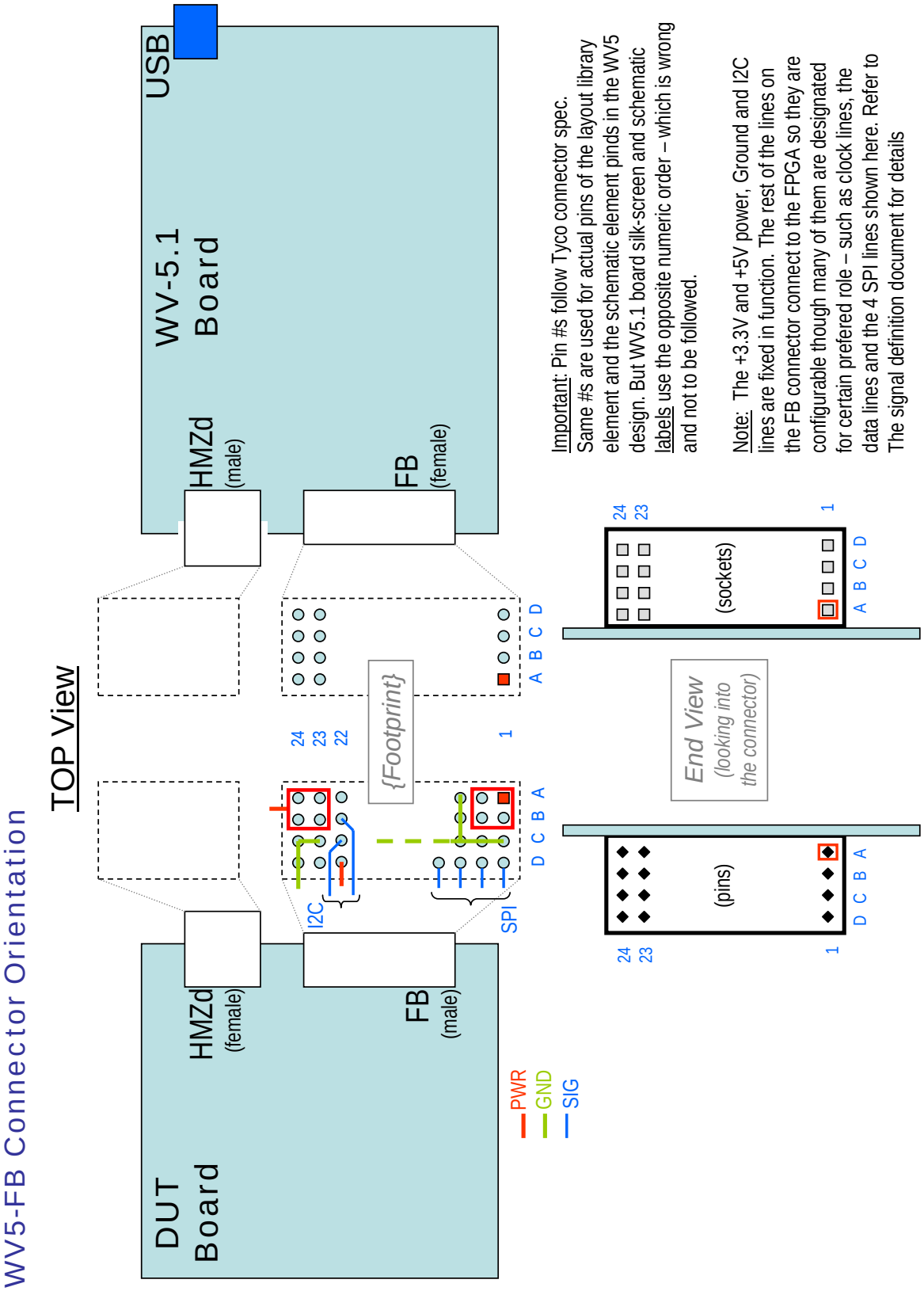


Figure 4: FB Connector orientation on the DUT board

5.1.2 HMZd Connector Orientation

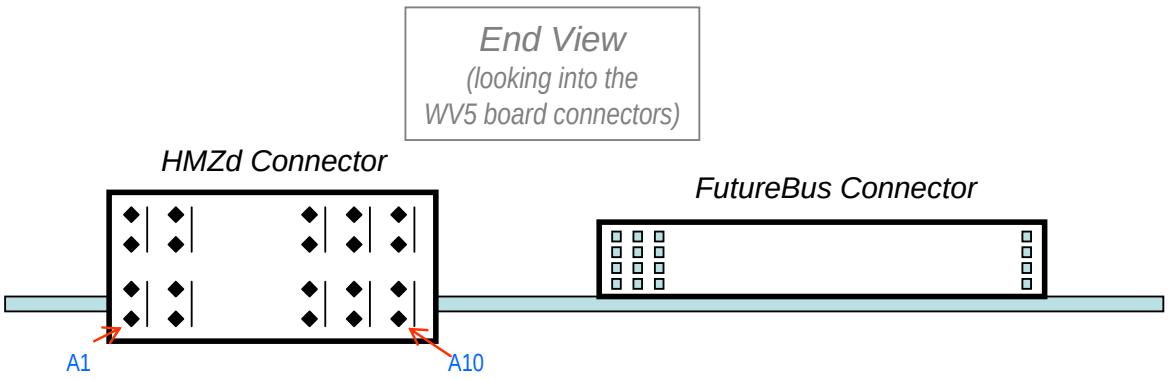


Figure 5: HMZd Connector Orientation on the DUT board

5.2 DUT Board Electricals

5.2.1 Power

The DUT board design is free to use local power supply or the +5.0V and +3.3V power supplied over the FB Connector and the +3.3V power supplied over the HMZd connector. The board identification EEPROM, which is required to be present on the DUT board, must be powered by the +3.3V power supplied by the WaveVision-5 capture board.

Following are the current limits of the WV5 connector power supplies:

FB +5.0V = 2.0 Amps
FB & HSP +3.3V = 500 mA

A DUT board may be designed with either or both of the two standard connectors - the FB (FutureBus) port and the HSP port - which is based on the HMZd connector.

5.2.2 FB Port

The FB Port uses the FutureBus 96-pin connector. The female side (receptacle) is installed on the DUT board. This port is used as the primary means for interfacing parallel data from a DUT or reference board to WV5 in the case of ADC-based channels; and in the other direction in the case of DAC-based channels. The interface provides up to 36 data/clock lines - four of which are especially designated for carrying source synchronous data clock. Thus one can implement two channels of up to 16 single-ended data signals each with a single-ended or differential clock signal. Alternatively, when higher speeds are required, a fully differential data path may be implemented with up to 16 differential (LVDS) pairs carrying data along with two differential (LVDS) clock signals. The port also provides several lower speed connections for control and status. An I2C bus is carried to connect with the DUT identification EEPROM required by the WaveVision system as well as to communicate with such devices as temp sensors. An SPI bus is also defined. DUT presence detect and power enable facilities are also provided for.

Other than the I2C bus and the DUT presence detect and power enable signals, all other signals of this port connect to the FPGA on the WaveVision-5 capture board. The table below specifies the functionality as well as the exact pin assignment of the signals. The DUT board designer is given the flexibility to provision the data, clock and general-purpose control signals as he wishes - thanks to the programmability provided by the WV5 FPGA. However, the power, ground, I2C and SPI signals must abide by the pin assignment shown in the table below. It is also highly recommended that the data strobe (clock) signal(s) be placed on the designated pins for best performance.

The data and clock signals on this port are capable of up to 200 mega-transfers/sec (100 MHz DDR) operation in CMOS mode and up to 500 mega-transfers/sec (250 MHz DDR) in the LVDS mode.

> Unused pins that are inputs to the WV5: shall be tied to ground if CMOS; shall be left floating if LVDS.
> Unused pins that are outputs of theWV5: shall be left open if CMOS; shall be terminated into 100-ohms if LVDS.

Refer to the *WV5 HSP and FB Port Specification* document for details of this port.

5.2.3 High Speed Port (HSP)

The HSP port uses the Tyco HMZd 60-pin connector. The female side (receptacle) is installed on the DUT/reference board; the male side is present on the WaveVision-5 capture board.

This port is intended to carry very high-speed signals that are often serial and almost always differential LVDS pairs. The interface provides up to 12 serial data lines in either direction and up to four serial clock lines in the signal-path to WV-5 direction (i.e., ADC application). In a DAC application, it is expected that the source-synchronous clock will be carried on the lines designated as data lines. When less than the maximum number of data or clock lines are employed on a given signal-path board design, the designer shall utilize the port starting from the pin A10 side and proceed in a contiguous manner.

+3.3V power, I2C bus, and four general-purpose I/O lines are provided. DUT presence detect and power enable facilities are also provided for. The power, ground, DUT power enable and the I2C bus must be located as shown in the table below. In addition, **if** an SPI bus is implemented over the general-purpose lines, then the assignment is recommended to be as follows: B7: SCSb, B8: SDI or SDI/O, B9: SDO or DIR, B10: SCLK. It is recommended that the SPI bus should be routed through the FB port if that port is utilized.

Finally, it is strongly recommended that the clock signals be carried over the pins indicated in the specification.

Refer to the *WV5 HSP and FB Port Specification* document for details of this port.

5.3 Identification EEPROM

A WV5-compliant DUT board shall include an identification EEPROM that complies with the requirements specified in the *SigPathData Interface Specification*.

6.0 Designing Hardware to Comply with WV5 Core

When designing a DUT or Reference Board that plugs into a digital controller card like the WaveVision-5.1, one must comply with the mechanical, electrical and functional requirements for these boards that are described in Section 5 of this document. However, there is another, higher level of board design one can do which includes the controller logic. In this case, the designer is interfacing directly with the WV5 Core. Referring to Figure 6, here you must comply with the SigPathData standard interface. Requirements of this interface are specified in its own document *(SigPathData Interface Specification)* - which is included here by reference.

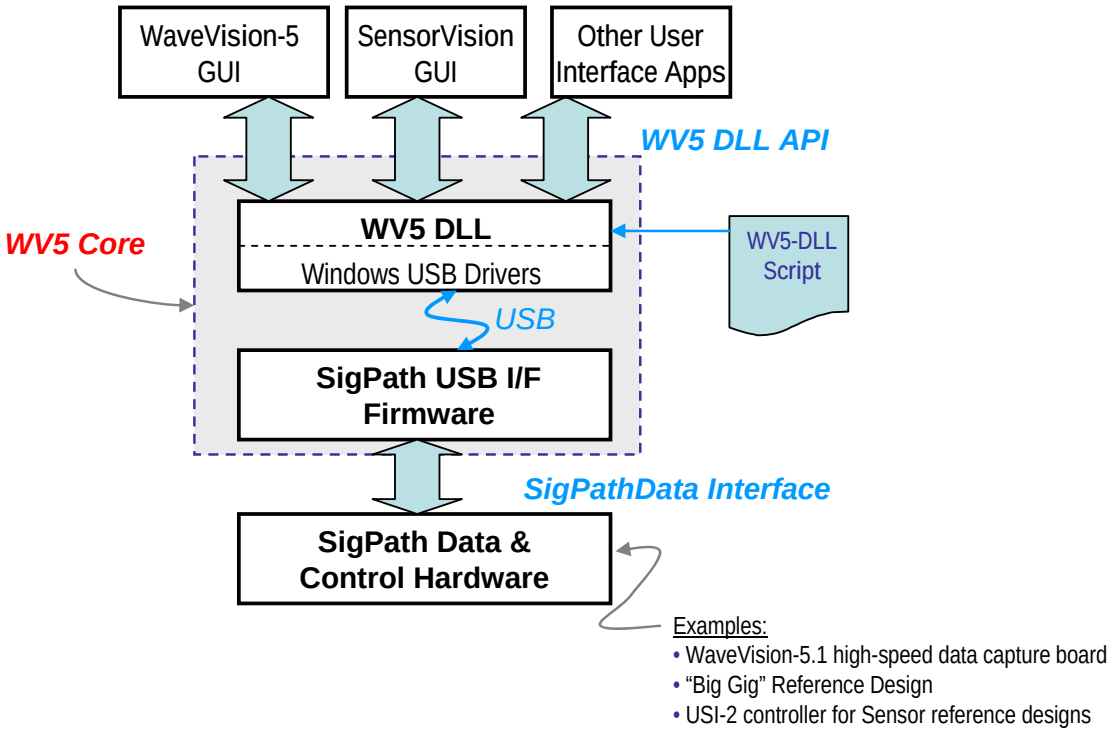


Figure 6: WaveVision5 High-Level Architecture

An essential part of making a DUT board compliant with the WV5 system is the inclusion of the identification EEPROM as this is the way the WV5 software discovers the connected board upon launch or power up. Section 7.1 describes how National engineers can program the EEPROM using the WaveVision software in a lab setting.

The external users of WaveVision Software are not given the ability to change/edit the contents of EEPROM. Therefore, the software by default does not expose its EEPROM editing capability.

The strict content/format requirements of the EEPROM are detailed in the SigPathData standard interface specification.

7.0 WaveVision5 As A Development Platform

The WaveVision5 software is primarily designed to be used by National's end customers. However, when a hardware developer is working with this system to debug the total package, certain additional capabilities can be enabled.

7.1 EEPROM Editor

The digital controller card EEPROM (a.k.a. Cypress EEPROM) and the DUT board EEPROM are essential for the workings of the WaveVision5 system. The WaveVision5 software includes a tool to program these EEPROMs. In order to enable this tool, do the following:

a. After having installed the WaveVision5 software, launch the application at least once. Then exit.

b. Go to the **C:\Program Files\National Semiconductor\WaveVision5\Data\Config** folder and open the file **WV5.ini**.

c. Under the [User Section], add a line: "EEPROM Editor=1".

d. Save and exit.

e. In the WaveVision5 software, under the Tools menu, the EEPROM Editor will now appear as a panel with two sections - one for the DUT EEPROM and the other for the Cypress EEPROM.

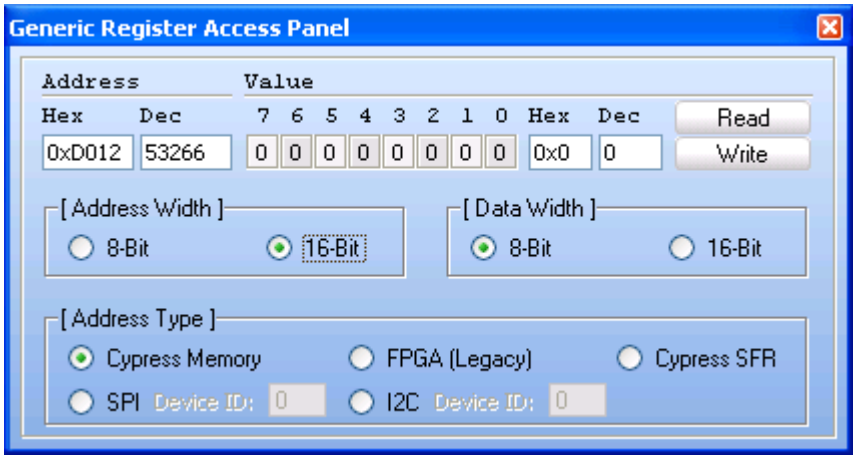
Note that the file used by this tool is a binary file. The contents of this file have to meet some precise requirements. These are specified in the SigPathData Interface spec.

7.2 Hardware Access Panel

The WaveVision5 system provides a basic utility to read/write the hardware memory space (which includes the FPGA registers) and the SPI registers on the signal-path devices. This is done through the Hardware Access Panel (also known at various previous times as the Debug Editor and Generic register Access Panel). It is available under the Tools menu. In case the panel does not appear under the Tools menu, make sure that there is a line **"Debug Editor=1"** in the **WV5.ini** file - as described in section 8.1.

Note that the Cypress Memory here means the memory space of the 8051 controller within the Cypress USB controller. Its address width is always 16-bit. The FPGA's registers are within this space starting at address D000h. Refer to the SigPathData Interface spec for more detail about the memory mapping.

The SPI access requires that the device SPI protocol comply with the National SPI-1.0 or SPI-1.1 spec. This is an internal spec that many signal-path devices use.



7.3 Working without the DUT EEPROM

There are two cases where a DUT EEPROM may not be present: integrated reference boards where the signal-path and the digital controller card functionality is on a single board; and during the development phase when the DUT/reference board does not yet exist but the developer has to debug his WV5 FPGA design. For both these case, the WaveVision5 system allows a method of specifying the DUT EEPROM with an equivalent file on the C: drive. This EEPROM-equivalent file is expected to be in the folder:

`C:\Program Files\National Semiconductor\WaveVision5\hardware\EEPROM`

Redirection to the EEPROM-equivalent file is done in the Cypress EEPROM - which must always be present. Note that if this redirection is present in the Cypress EEPROM, it will supersede even if a DUT/Reference board is later attached to the WaveVision5 controller board and a physical DUT EEPROM is now present.

In order to redirect the DUT EEPROM search to the equivalent file on C: drive, do the following:

a. Connect the digital controller board (e.g., WaveVision5, USI-2 etc.) and launch the WaveVision5 software.

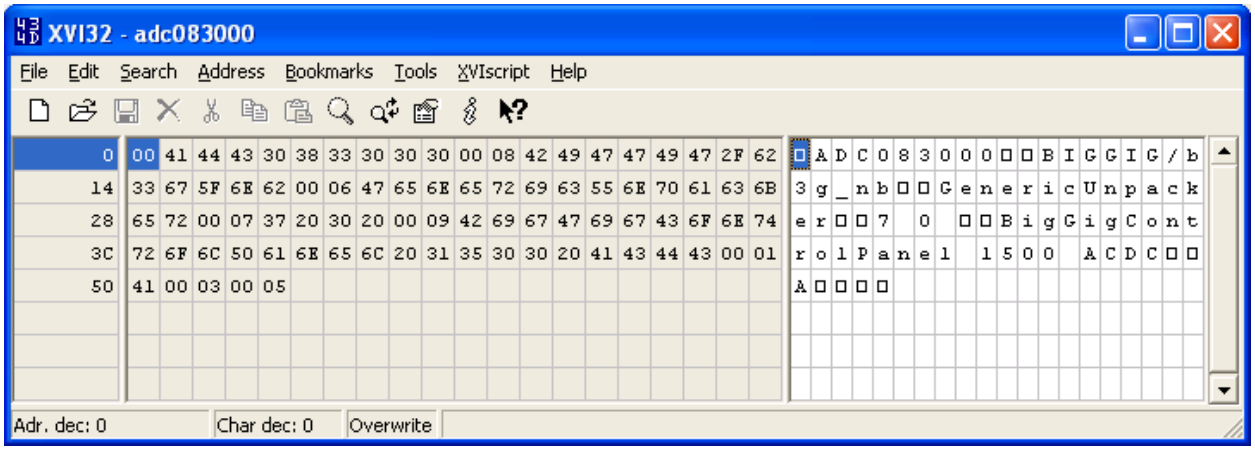
b. Power-on the board. The software will discover it and attempt to configure it. Not finding the DUT/Reference board EEPROM, it will load a basic WV5 FPGA image.

c. Go to TOOLS -> EEPROM Editor.
The program will complain that it "Can't read DUT EEPROM....". Say OK (twice).

d. EEPROM Editor panel will appear. The Cypress EEPROM's contents are displayed on the left side. At the bottom of this section, there is a field called "DUT EEPROM FILE". For the value of this field, enter an alphanumeric string that will be the name of your EEPROM file. For example, you may use the name "redirectEXP1".

e. Click on the PROGRAM button on the left half of the panel. The Cypress EEPROM is now programmed to redirect the search for the DUT EEPROM to a file. Exit the panel and the WaveVision5 program.

Next you must provide the EEPROM-equivalent file for your project. Under the EEPROM folder mentioned above, there are several pre-existing EEPROM binary files. Open one of these with a binary editor program such as the **`XVI32.exe`**.



Change only the first two fields - e.g., the ones containing "ADC083000" and "BIGGIG/b3g_nb" in the above example. The first of these is the precise name of your project. The second is a descriptive name that you may optionally change. Let's say the name you have chosen for your project is "EXP1". Enter this character string in the first field (in place of the "ADC083000" shown in the figure above). Save this binary file with the exact name that you entered in the Cypress EEPROM in step (d) above - for example "redirectEXP1".

The WV5 Core software, upon next launch, will now configure itself and the FPGA on the digital controller board with files whose name contains the string that you specified as the precise name for your project. In this example, since you named your project EXP1, the FPGA image file name in the hardware\fpga_images folder must be: **`wv5_xc4vlx25_EXP1.bit.`**

As you know, in addition to the FPGA image file, WV5 also needs to use a proper script file. Your script file should also have a name based on the DUT name of "EXP1" that you chose above. If you have a script file for your project, then name it appropriately in the appropriate directory (see the Directory Structure section in this document and the *WV5 Core Naming Convention* document). If you do not yet have a custom script written for your project and you just want to do basic read/write access on the capture hardware, then you should just use one of the basic, generic script files that already exist in the WV5 directory. A recommended script file is "`wv5_default_2.cap.xml`". Create a copy of this file and name it "**`Wv5_xc4vlx25_exp1.dut.xml`**" for this example case.

(Note: Even if no specific script file is provided as described above, the WV5 Core software will still default to a set of script files and provide basic functionality).

8.0 WV5 Core Software

The WaveVision5 system allows you to modify or develop associated software at two levels:

(a) certain aspects of the user interface application (like the WaveVision-5 GUI) and hardware control (like DUT configuration) can be customized to for each new board using the Script utility.

(b) new user interface applications can be written that meet your specific needs.

Refer to Figure 6 for the relationship of applications and scripts with the WV5 Core software.

8.1 Directory Structure of the WV5 Core software

```
C:\Program Files\National Semiconductor\WaveVision5\wvdll.dll
C:\Program Files\National Semiconductor\WaveVision5\libcint.dll
            DLL to support C scripting

C:\Program Files\National Semiconductor\WaveVision5\hardware\eeprom_images
            EEPROM images for all-in-one boards

C:\Program Files\National Semiconductor\WaveVision5\hardware\firmware_images
            *.bix files for Cypress firmware

C:\Program Files\National Semiconductor\WaveVision5\hardware\fpga_images
            *.bit files for FPGA images

C:\Program Files\National Semiconductor\WaveVision5\hardware\scripts
            *.xml files for XML scripts
            *.{cpp,h} for C scripts
```

8.2 Naming convention

It is essential for the engineer wanting to customize the WV5 software to first gain familiarity with the naming convention used in this system. Please refer to the *WV5 Core Naming Convention* document.

As mentioned elsewhere, the identification EEPROM on the DUT board provides the identifier that is used by the WV5 Core to configure itself with the appropriate files for the board currently connected. Over time, a method has been developed to handle exceptions to this basic scheme. These exceptions are handled through a mapping file named **"image_map.xml"**. The document *WV5 Core image_map Spec* describes this method.

8.3 Working in Simulator Mode

A basic utility is provided in WV5 developer environment to allow you to develop a new script or even a new application without the corresponding hardware. For example, while a new DUT board is being fabricated and assembled, the application engineer may want to develop his custom register control panel. The WV5 Simulator provides the means to do this.

IMPORTANT NOTE: The simulator is not to be used simultaneously with real hardware. The simulator will only activate if the DLL starts with no physical USB device attached. If a physical USB device is attached, then the simulator is closed down and removed and the physical USB device is started.

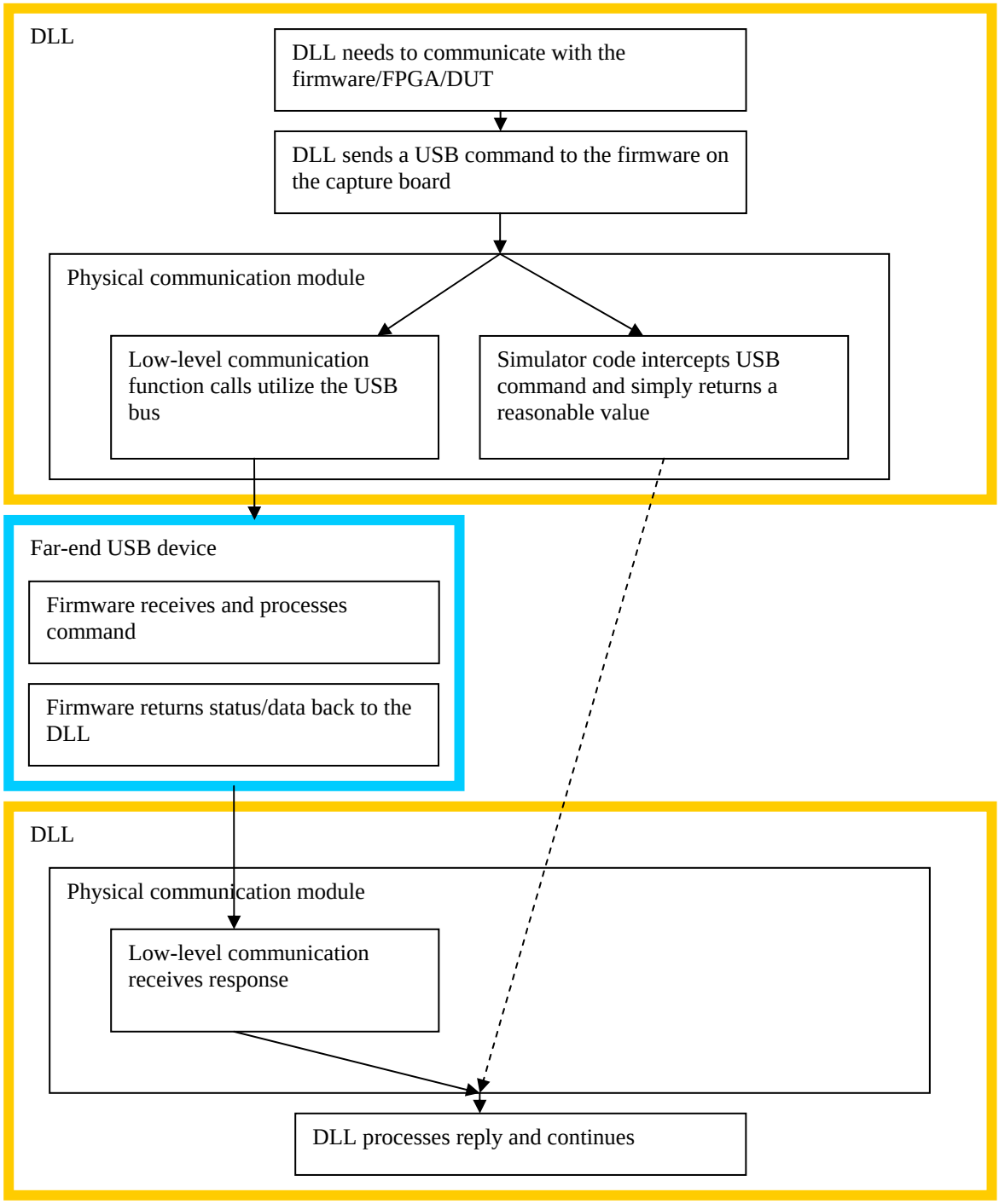
8.3.1 Enabling the Simulator

The simulator is enabled by the presence of a file called "**__sim.xml**" in the hardware\scripts directory. The file name is two underscore characters (__) followed by "sim.xml". This particular name was chosen so that when the hardware\scripts directory is sorted by name, this file, if it exists, will be the very first entry.

Since the existence of the __sim.xml file enables the simulator, this file is not included in the default distribution. This file must be created by the user.

8.3.2 Implementation of the Simulator

The simulator simulates the low level USB calls. Usually, when the DLL needs to communicate with the firmware, FPGA, or DUT, it uses the USB bus to send commands to the firmware running on the capture board. In the case of the simulator, all the USB communication is simulated to return some reasonable values.



The dotted line represents that fact that there is no USB communication; however the rest of the DLL is convinced some USB communication occurred and that the firmware returned some value.

So, only a small part (the physical communication module) of the DLL is aware a simulator is running. All other parts behave exactly the same regardless of simulation or not. The benefit of this is that many other modules of the DLL can be tested without any hardware.

8.3.3 Reasonable Return Value

To simulate every single DUT to exact hardware specifications would be extremely difficult. The simulator implemented in the DLL does not attempt anything of the sort. Basically, the simulator interprets the command and always returns a "true" or "success" status; there are no error checks.

When data is to be read, for instance data from a register, the simulator returns a random value. This includes data for captures; so the simulator can process capture requests, but the data returned will be complete garbage.

8.3.4 Contents of __sim.xml

The file is a very small XML file that has the following structure:

```
<wv5>
    <sim list_of_simulated_params />
</wv5>
```

The key to the file is the "sim" element. Below are the parameters that can be simulated:

dut_desc

The value for this attribute is the DUT description usually found in the DUT EEPROM.

data_params

The value for this attribute is the data parameter field usually found in the DUT EEPROM.

dut_eeprom_file

This is to simulate the "DUT EEPROM File" entry in the Cypress EEPROM. This field is used when the capture board does not support a DUT EEPROM.

fpga_clock_count

The number of clock counts to simulate when attempting to calculate the clock frequency.

interface_id

The value for this attribute is the interface ID of the FPGA.

vid

The value for this attribute is the USB VID number, usually used for capture board identification.

pid

The value for this attribute is the USB PID number, usually used for capture board identification

board_desc

The value for this attribute is the board description usually found in the Cypress EEPROM.

fpga_image

FPGA image name. This field is called "Base Firmware" in the WV5 GUI EEPROM editor.

8.3.5 Examples

Simulate WV5 capture board + ADC14DS105

```
<wv5>
    <sim vid="0x0400" pid="0x2007" dut_desc="adc14ds105" data_params="13 0 29
16" />
</wv5>
```

The VID and PID fields specify the WV5 capture board, and the rest describe the AD14DS105.

Simulate WV5 capture board + SP1250MI02

```
<wv5>
    <sim vid="0x0400" pid="0x2007" dut_desc="spi1250mi02" data_params="11 0 27
16" />
</wv5>
```

Simulate ADC10D1000

```
<wv5>
    <sim vid="0x0400" pid="0x200f" dut_eeprom_file="adc10d1000acdc" />
</wv5>
```

Notice in this case there is no need to fill in the "dut_desc" field or the "data_params" field. The reason is the DUT EEPROM image file ("adc10d1000acdc") already contains that information. Remember that the all-in-one boards like the ADC10D1000 already simulate the DUT EEPROM by having the DUT EEPROM image in the eeprom directory on the PC.

8.3.6 Simulating a non-existent DUT

The simulator can be used to simulate both currently supported DUTs and DUTs that do not exist yet. If the simulator is enabled for an existing DUT, then all of the other support files like the FPGA image and script files will already exist.

However, for non-existent DUTs, all of these supports files must be created. The script file will need to be created in the hardware\scripts directory using the standard naming convention. And a corresponding FPGA file will need to be created in the hardware\fpga_images directoy.

The FPGA image can be empty but it does need to exist. The filename of the .bit file should follow the naming convention.

8.4 WV5 Data Model

Figures 7a-c show the conceptual model of data flow from the analog front-end through the FPGA/controller card buffer to the PC memory buffer where the data is finally delivered to the application by the WV5 Core software.

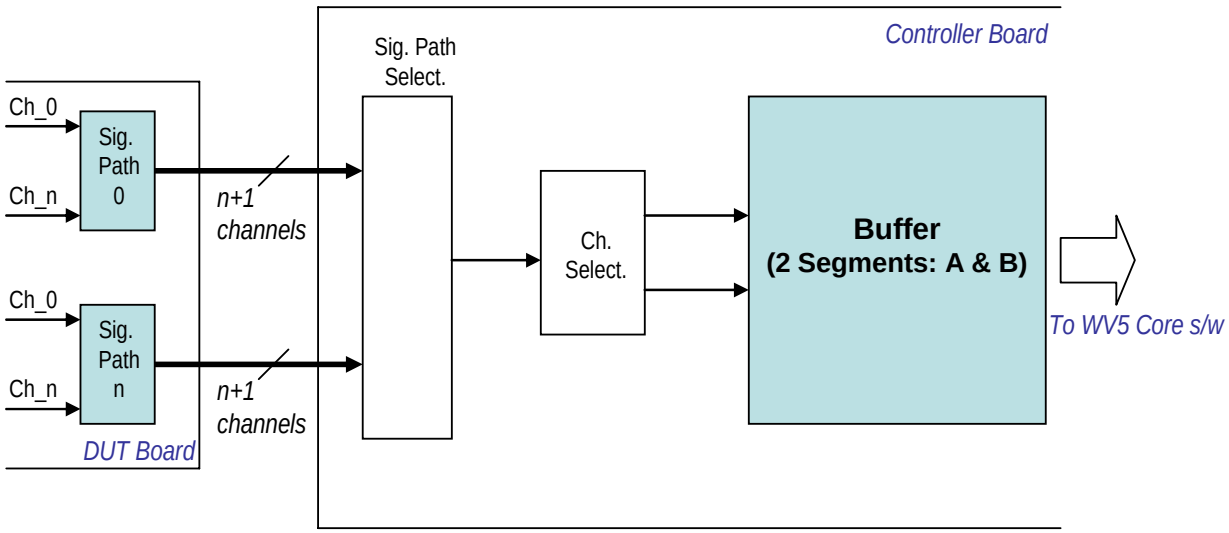


Figure 7a: Signal Paths, Channels and the Capture Buffer.

The visibility of the WV5 Core software extends to just the Buffer in the controller card. Figure 7a shows how the analog front-end data gets into the buffer. Figures 7b and 7c show how the WV5 Core software then treats the data from thereon.

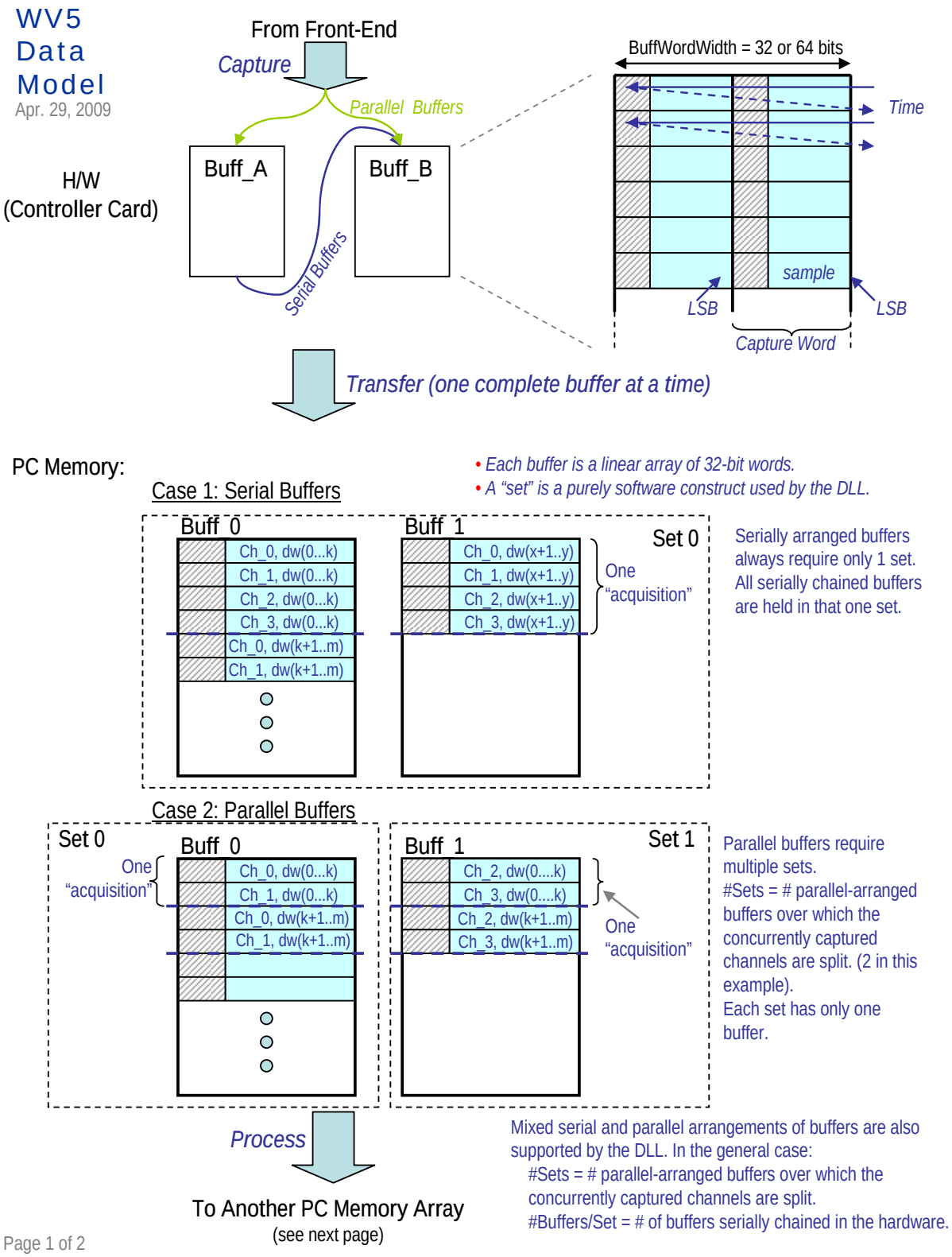


Figure 7b: WV5 Data Model - Part 1

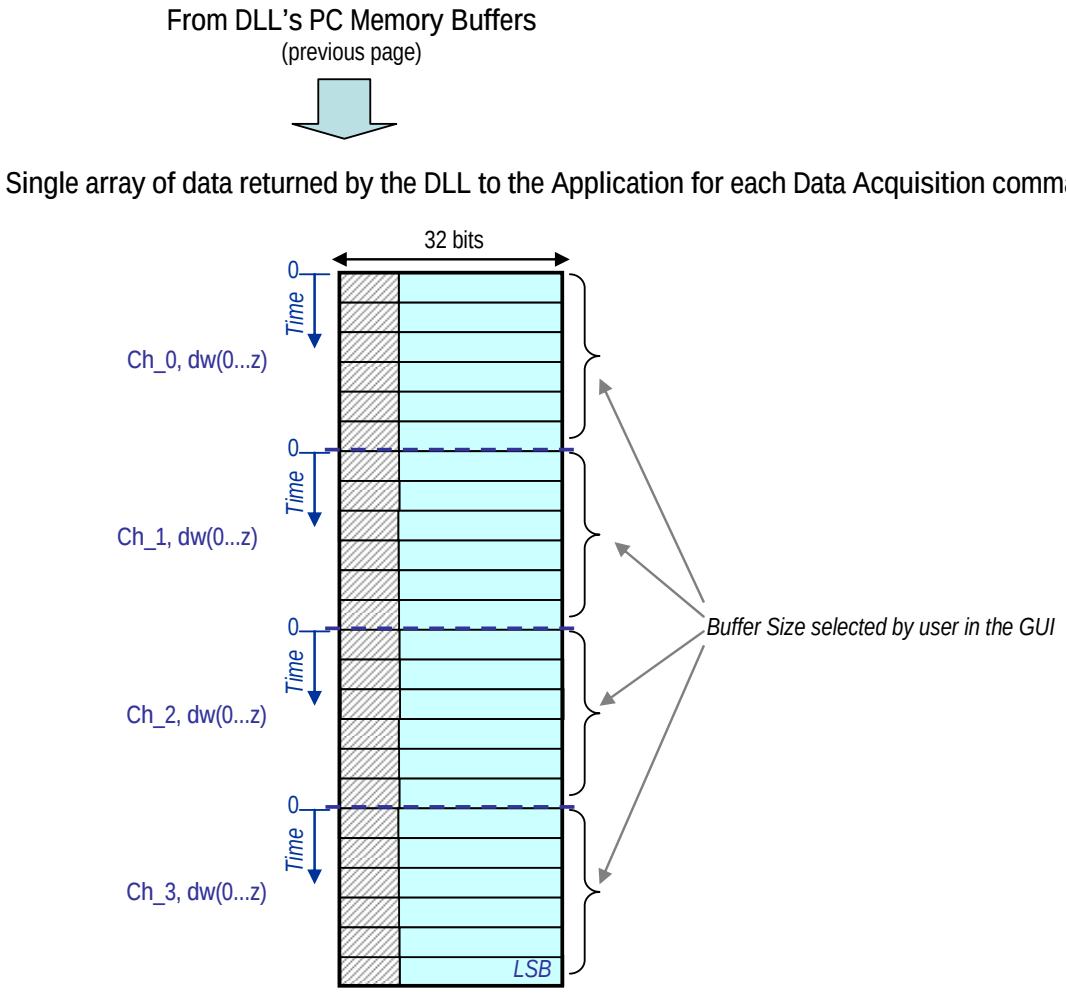


Figure 7c: WV5 Data Model - Part 2

Refering to Figure 7b and 7c, the following terms define the constructs that the WV5 software uses. These terms are used in the Script documentation.

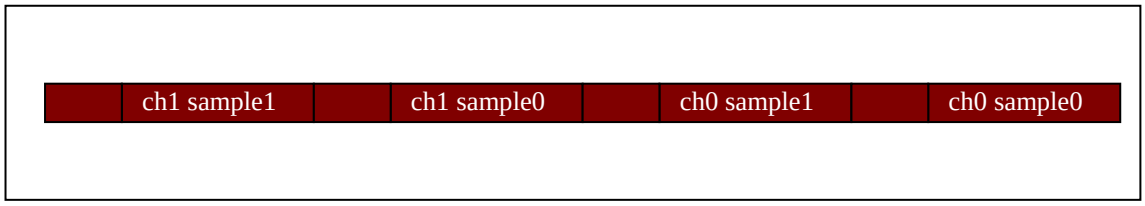
Acquisition

An acquisition is a construct that only exists in the DLL. The firmware only deals with bytes and FPGA words. The user of the DLL only deals with samples.

The DLL uses acquisitions to group several FPGA words together to help in de-interleaving captured data. Note: since sets only contain one type of acquisition, the DLL cannot de-interleave data that span sets.

One acquisition may contain several samples from several channels. The location of each sample within an acquisition is determined by a list of positions where the LSB of the sample is found.

Below is an example of an acquisition that contains two samples from two channels. Assume each sample is 10 bits and each sample starts in a 16-bit boundary.

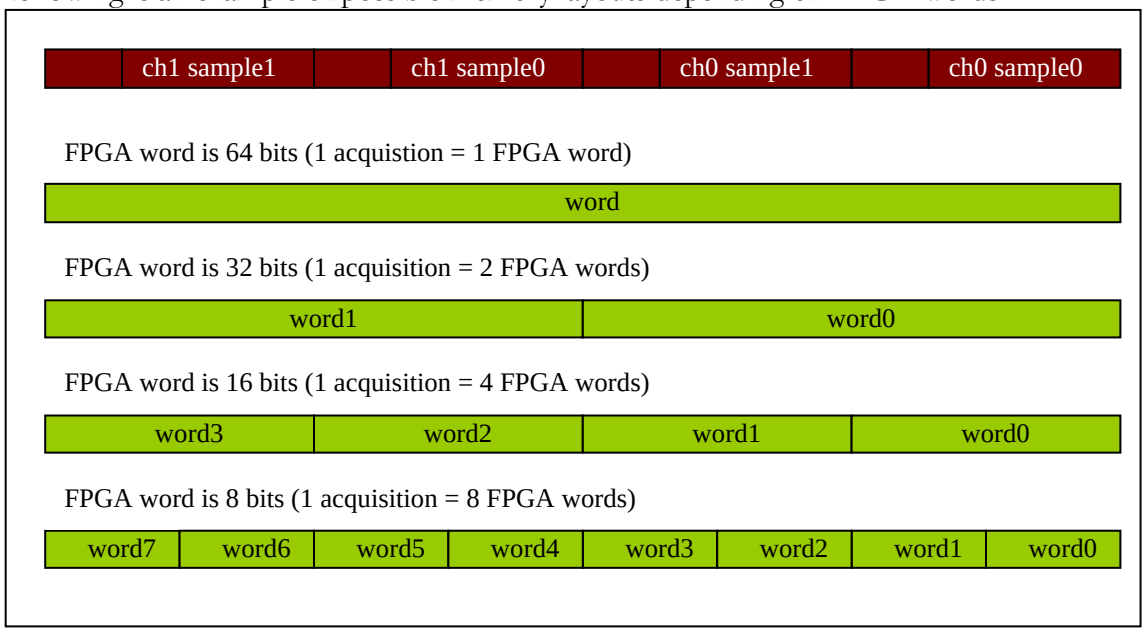


Each sample would be 10 bits wide and the blank boxes would be 6 bit wide.

In this example, the list of bit positions would look like:

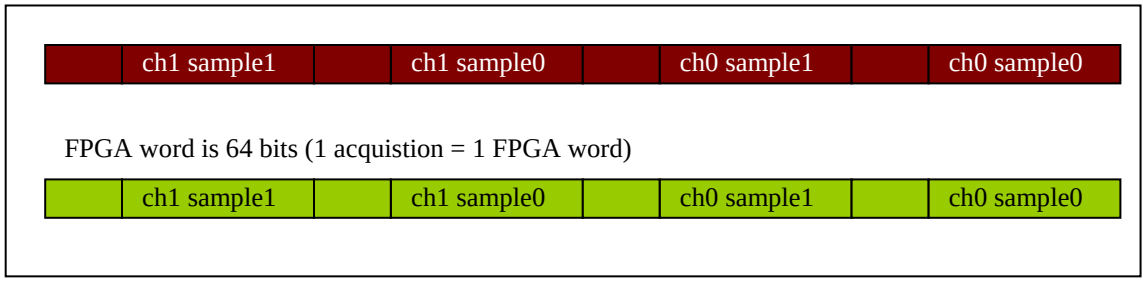
LSB bit position for ch0, sample0: 0
LSB bit position for ch0, sample1: 16
LSB bit position for ch0, sample0: 32
LSB bit position for ch2, sample1: 48

The following is an example of possible memory layouts depending on FPGA words:



The main goal of the acquisition is to allow the DLL to process data that is interleaved across FPGA words.

Most of the time, the acquisition is one FPGA word. So, using the current sample:



So most of the time, both acquisition and FPGA word look the same. The major limitation of the acquisition is that the DLL requires that all the samples from one channel be grouped together.

All the buffers in one set must contain the same type of acquisition. Otherwise the code in the DLL that processes the data for use by the client application will not properly work.

Set

Just like an acquisitions, a set is a construct that exists only within the DLL. A set is used to collect all buffers that share the same channel source. If buffers do not share the same source channels, then those buffers must be placed into different sets.

By keeping these buffers separate, the DLL can more easily de-interleave the data to be returned to the client of the DLL.

8.5 Customizing Using Script Files

All boards that connect to the WV5 Core software must have two script files each: the board script file and the DUT script file. By name, this pair of script files is unique to each supported board (assuming each board has a unique name as specified in the identifier EEPROM). In addition, there is a common DLL/Script API file that all boards also require to be present.

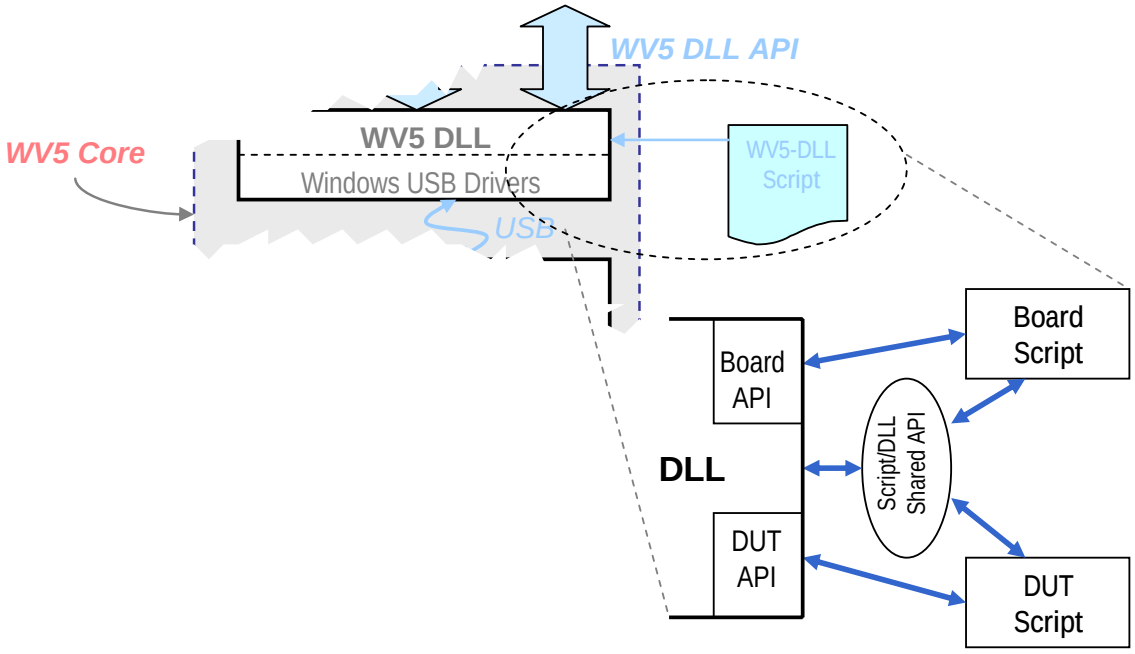


Figure 8: DLL to Script File Interface

Note that the term "board" here refers to the controller (a.k.a. capture) board and the term "DUT" refers to the DUT/reference board.

Both the board API and DUT API requires that the script files implement a set of mandatory functions. These mandatory functions have very specific names. These names are generated as a combination of the file name and a function name.

8.5.1 Board API

The functions are grouped into two categories, initialization and generic hardware access. The initialization functions initialize the internal data structures and the hardware. The general access functions are called when the user of the DLL requests a read or write to/from the hardware. These general accesses are not tied to any specific DLL action. For example these are called when the "Hardware access panel" is used on the WV5 GUI.

8.5.2 DUT API

These functions are grouped into five categories: initialization, GUI, ADC capture, DAC download, and generic hardware access.

8.5.3 Script/DLL Shared API

There is one final API. This API is a general set of functions shared by both the DLL and scripts to pass data back and forth.

8.5.4 Script Naming Convention

The naming convention relies heavily on image_map.xml file. Please first familiarize yourself with the WV5 Core image_map Specification.

Board script filename

All boards have a nickname assigned to it in image_map.xml. This nickname is the basis for the board script file and the extension "brd.cpp" is simply added at the end:

VID	PID	nick	Board script name
0x0400	0x2009	biggig.2.0	biggig.2.0.brd.cpp
0x0400	0x2007	wv5	wv5.brd.cpp
0x0400	0x2010	spio5.5	spio5.5.brd.cpp

DUT script filename

The name for the DUT script also uses image_map.xml. There are several fields that can be used in generating the DUT script name. In general, if the DUT entry has an exception in image_map.xml, then the name given in the override is used as the basename. Otherwise, the script basename is generated by concatenation the board nickname, the FPGA part ID, and the DUT name all with the underscore ("_") character. Finally, ".dut.cpp" is appended to the basename:

DUT	Override	VID	PID	Nick	FPGA Part ID	DUT scipt name
spio5.5	spio5.5_xc5vsx95t_dac					spio5.5_xc5vsx95t_dac.dut.cpp
exdut1		0x0400	0x2007	wv5	xc4vlx25	wv5_xc4vlx25_exdut1.dut.cpp

Mandatory Function Names for the Board API

The DLL calls the board script functions by generating a name derived from the script filename. The formula is the script name without the trailing .brd.cpp, appended with an underscore ("_"), and then appended with the base function name.

This generated name is then processed to replace all periods (".") with underscores ("_"). This is done because the period is character with special meaning in C.

Board Script name	Remove ".brd.cpp" (result is same as nick)	Function	Append all parts	Final function name
biggig.2.0.brd.cpp	biggig.2.0	create	biggig.2.0_ create	biggig_2_0_ create
spio5.5.brd.cpp	spio5.5	create	spio5.5_create	spio5_5_create
wv5.brd.cpp	wv5	poll_hook	wv5_poll_hook	wv5_poll_hook

Mandatory function names for the DUT API

This operates exactly the same was as the board API naming convention. The script name is used, the ".dut.cpp" is removed, an underscore is appended, the function name is appended, and finally all periods are replaced with underscores.

DUT Script name	Remove " .dut.cpp"	Function	Append all parts	Final function name
spio5.5_xc5vsx95t_dac.dut.cpp	spio5.5_xc5vsx95t_dac	create	spio5.5_xc5vsx95t_dac_create	spio5_5_xc5vsx95t_dac_create
wv5_xc4vlx25_exdut1.dut.cpp	wv5_xc4vlx25_exdut1	poll_hook	wv5_xc4vlx25_exdut1_poll_hook	wv5_xc4vlx25_exdut1_poll_hook

8.5.5 Summary of API Functions

{*The document* ***Programmers' Guide to WV5 C-Script*** *provides a complete technical reference*}.

Most of the functions in both the board API and the script API take one parameter. This parameter is a pointer to the script-specific data. The script may or may not use script specific data. If it does, the script needs to create an instance for each physical board or DUT. This is necessary so that multiple devices can exist simultaneously.

Board API

Please consult the technical documentation or **`script_api.h`**. Either of these documents will describe **`brdprefix.`**

The mandatory functions are:

```
Interface_Ret brdprefix_version();
Custom_Brd_Data* brdprefix_create();
Interface_Ret brdprefix_destroy(Custom_Brd_Data* board_data);
Interface_Ret brdprefix_execute_board_initialization(Custom_Brd_Data* board_data);
Interface_Ret brdprefix_poll_hook(Custom_Brd_Data* board_data);
Interface_Ret brdprefix_general_access_rd(
    Custom_Brd_Data*       board_data,
    Wv_Debug_Access_Method access_method,
    WvWord                 bus_index,
    WvWord                 device_address,
    WvWord                 address_bitwidth,
    WvWord                 address,
    WvWord                 data_bitwidth,
    WvWord*                rd_val
    );
Interface_Ret brdprefix_general_access_wr(
    Custom_Brd_Data*       board_data,
    Wv_Debug_Access_Method access_method,
    WvWord                 bus_index,
    WvWord                 device_address,
    WvWord                 address_bitwidth,
    WvWord                 address,
    WvWord                 data_bitwidth,
    WvWord                 wr_val
    );
```

DUT API

Please consult the technical documentation or **`script_api.h`**. Either of these documents will describe **`dutprefix`**.

The mandatory functions are:

```
Interface_Ret dutprefix_version();
Custom_Dut_Data* dutprefix_create();
Interface_Ret dutprefix_destroy(Custom_Dut_Data* dut_data);
Interface_Ret dutprefix_execute_dut_initialization(Custom_Dut_Data* dut_data);
Interface_Ret dutprefix_get_clock_frequency(Custom_Dut_Data* dut_data);
Interface_Ret dutprefix_poll_hook(Custom_Dut_Data* dut_data);
Interface_Ret dutprefix_gui_tab_enum(Custom_Dut_Data* dut_data);
Interface_Ret dutprefix_gui_control_enum(Custom_Dut_Data* dut_data);
Interface_Ret dutprefix_gui_control_change(Custom_Dut_Data* dut_data);
```

```
Interface_Ret dutprefix_get_capture_parameters(dutprefix_DataCustom_Dut_Data
dut_data);
Interface_Ret dutprefix_execute_capture_actions(Custom_Dut_Data* dut_data);
Interface_Ret dutprefix_execute_post_capture_actions(Custom_Dut_Data* dut_data,
WvWord* captured_data);
Interface_Ret dutprefix_get_dac_download_parameters(Custom_Dut_Data* data);
Interface_Ret dutprefix_dac_cmd_process(Custom_Dut_Data* dut_data);
Interface_Ret dutprefix_general_access_rd(Custom_Dut_Data* dut_data);
Interface_Ret dutprefix_general_access_wr(Custom_Dut_Data* dut_data);
```

Script/DLL Shared API

The script and the DLL need to share data. Most of time, the data is either function parameters or function return values. To do this, a separate API is used.

The shared API is also part of the technical documentation.

8.5.6 Difference between Board/DUT API and Shared API

The main difference between the APIs is that the board/DUT API is dynamic meaning the function names change depending on the physical board/DUT, but the shared API is static meaning the function names do not change.

Although the dynamic names allows for arbitrary device names (which eventually determines the script names which determines the prefix used on the function names), one limitation is that data cannot be shared between DLL universe and the script universe when using these functions.

For instance, assume the DLL declares an integer variable x. And assume one of the script files implements a function that is declared "void test_dut_api_function(int function_argument)".

Since "test_dut_api_function" is a function name that was created dynamically, there is no way for the DLL to say "test_dut_api_function(x)" because the DLL has already been compiled.

However, using the static API, the DLL can use its data. So, assume there was a shared function declared "void shared_api_function(int function_argument)". The DLL can call "shared_api_function(x)".

And the static functions can also be used by the script. And it also has access to the same data the DLL has.

This idea is to use this API to pass actual function parameters between the DLL and the script.

The chronology is:

1) User of DLL requests an action that involves invoking a script function
2) DLL preps for calling script function
3) DLL calls a function in the shard API to set all the function parameters
4) Shared API stores the function parameters
5) DLL calls a function in the dynamic API to activate the script function
6) Script function calls shared API and requests all the function parameters
7) Shared API passes the function parameters back to the script
8) Script is now ready with all the parameters to execute the main body of the function.

9) Script executes the body of the function
10) Script passes any return values to shared API
11) Shared API stores return values
12) Script function exits
13) DLL calls the shared API to retrieve the return values

Example of using shared API in the script

This example highlights steps 6 – 12 from the above chronology.

```
Interface_Ret test_dut_gui_control_enum(Test_Dut_Data* data) {
    WvWord tab_index;
    Interface_Ret ret;

    // Print some debug text to the log file
    script_debug_out_pre("Entering test_dut_gui_control_enum");
    script_debug_out_endl();

    // Use the shared API and retrieve the tab number.  Make
    // sure the return code is checked before using the value
    ret = script_get_params_gui_control_enum(&tab_index);
    if (ret != INTERFACE_RET_OK) {
        // Error.  Make sure the error is logged.
        return script_process_return(
            ret,
            "test_dut_gui_control_enum"
            );
    }

    // Tab index is ok.  Check the range.  Let's just say this test GUI
    // had only 1 tab
    if (tab_index != 0) {
        // tab_index out of range
        return script_process_return(
            INTERFACE_RET_ERROR,
            "test_dut_gui_control_enum: tab index out of range"
            );
    }

    // ... and let's assume this tab has 1 GUI object and that
    // GUI object was stored in the custom data structure.
    script_set_return_gui_control_enum(1, data->gui_object);

    return script_process_return(
        INTERFACE_RET_OK,
        "test_dut_gui_control_enum"
        );
}
```

8.6 Writing User Interface Applications

The complete WaveVision-5 system includes hardware (e.g., the WaveVision-5.1 controller card); the low-level firmware and software that make up the WV5 Core; and the WaveVision-5 Data Acquisition and Analysis software (see Figure 6). Due to the virtue of the WV5_DLL API, it is possible to write other user interface GUI applications than the base WaveVision-5 GUI. These can vary from the very simple control panels to much more sophisticated GUIs. But the motivation to do so would be to customize the user interaction with the hardware beyond what the scripting utility provides.

It should be noted that the WV5_DLL API is primarily providing an interface for an application program that is presumed to be written in C/C++ to run in a Windows environment. Applications written in other languages such as C# would require additional effort in interfacing the two programs.

A separate document, **Programmer's Guide to WV5_DLL API**, has been written to assist the application developer.

8.7 Adding a New Board to the System

Occasionally, support for a new board must be added to the system. There are several steps necessary to get the software to recognize the board.

8.7.1 Acquiring a new VID/PID

The first step does not involve software but rather some paperwork. The new board must have a unique VID and PID numbers. These numbers are fundamental to the software. These numbers are control by the intellectual property group at National Semiconductor. It is the responsibility of the board developer to acquire a new VID/PID combination.

The VID number is assigned by the USB governing body and all National Semiconductor products use a VID number of 0x0400. The PID must be acquired from the IP group.

8.7.2 Plugging in the New Board for the First Time (Setting up Windows)

The Cypress EEPROM will not programmed with any reasonable value straight from the factory. The Cypress microcontroller will detect this and identify itself as a Cypress product to Windows, even though the board is produced by National Semiconductor.

To solve this, the WV5 GUI will be used to program the Cypress EEPROM. This will convince the Cypress microcontroller and consequently Windows that the hardware is produced by National Semiconductor.

So, the first step is to make sure the WV5 GUI is installed.

Then plug in the board.

At this point, Windows should bring up a dialog box with the message "Welcome to the Found New Hardware Wizard". Choose "No, not this time". Click Next. Choose "Install the software automatically". Click Next. Let Windows find the driver and install it. At the "Completing the Found New HardwareWizard" dialog box, click Finish.

If this does not succeed, the driver may have to be choosen manually. The relevant files are:

%WINDIR%\inf\wv5.inf
%WINDIR%\system32\drivers\wv5.sys

To check that the driver was installed properly, open up the "Device Manager" control panel and open up expand "Universal Serial Bus controllers". There should be an entry "National Semiconductor Unprogrammed board".

Start the VW5 GUI and open up the EEPROM Editor. In the "[Cypress]" section fill in the appropriate VID and PID numbers and click program.

Close WV5.

Unplug the board.

Now, Windows must be told that the new VID/PID combination should be supported by the National Semiconductor software. To do this, %WINDIR%\inf\wv5.inf needs to be edited.

There are only two places that need to be modified. The first is in the section labeled "[National]". There is a list of currently supported boards. Follow the pattern and create an entry using the new VID/PID combination.

The second section is near the bottom under the "[Strings]" block. Again follow the pattern and add a new entry for the new VID/PID. Please remember to add a descriptive name. Save this file. Then right click on the file and choose "Install".

Plug the board back in.

Windows should once again bring up the "Found New Hardware" wizard. This time, Windows is attempting to find the proper driver for the new VID/PID combination. As before, do not check for updates and let Windows chose the software automatically.

To double check that Windows has successfully linked the VID/PID with the National Semiconductor driver, open up the "Device Manager" and expand "Universal Serial Bus controllers". There should be a new entry with the name choosen in the "[Strings]" section of the INF file.

8.7.3 Adding a New Entry to image_map.xml

Windows is now ready. The next step is to prep the WV5 software. To do this image_map.xml needs to be edited. In the <board_properties> section, there are several entries for all the different boards selected. Choose a board that has the similar properties as the new board and then copy and paste and edit the line to suit the new board.

All of the parameters in the <board> element are describe in the "WV5 Core image_map Spec" document.

8.7.4 Adding New Firmware

The last preparation step is preparing the firmware image. Assuming custom firmware is not necessary, then this will simply involve copying and renaming an existing firmware image.

Open up the firmware_images directory. Find a firmware image is most closely related to the new board. Copy that image file and rename it assuming the file naming convention.

At this point, the new board is ready for basic communication. This means that the WV5 GUI can now be started and the EEPROM Editor can be used to customize the EEPROMs.

Any additional functionality will require creation of custom board and DUT scripts.

8.8 Patches

There are occasional updates to the core software. For minor updates, a patch executable is distributed.

8.8.1 Patch installation

To install the patch, copy the patch executable into the WV5 GUI installation directory. And then run the patch executable. The patch installer simply overwrites the existing file with the new files.

Associated with each patch is a text file that describes the contents of the patch. This is named _patch_rev.txt.

To view the contents of the patch and contents of _patch_rev.txt, the patch executable can simply be run in a separate directory.

Example:

Assume a new patch file called new_patch.exe. To see the contents:

- Create a new directory called c:\temp\patch
- Copy new_patch.exe to c:\temp\patch
- Run c:\temp\patch\new_patch.exe
- Use notepad to view c:\temp\patch_patch_rev.txt

Appendix A- DUT EEPROM field in Cypress EEPROM

There is a field called the "DUT EEPROM field" in the Cypress EEPROM. If this is filled in, then the DLL uses the value as a filename and looks for that file in the eeprom subdirectory.

This method was originally used for the BigGig boards. These boards were "all-in-one" boards which contained both the Cypress and the DUT. However, these boards do not have a DUT EEPROM. To determine the actual DUT being tested, this "DUT EEPROM field" was added to the Cypress EEPROM so that the DLL can look for the file locally on the computer.

This file on the computer is an exact image of a real DUT EEPROM. This means these are binary files that contain the same contents as a real DUT EEPROM. To decode the contents of the DUT EEPROM, please refer to the eeprom documentation.

This field should be used in only two cases:

1) The capture board is in fact a capture + DUT board combined into one. In these cases, the DUT doesn't change and so the DUT EEPROM doesn't change, and so the use of a static file is valid.
2) A DUT board is being debugged and the DUT EEPROM has not been programmed yet. In this case, the engineer can force the DLL to use a particular DUT EEPROM image located on the computer.

SensorPath specific comments:
The very first SensorPath board was an "all-in-one" board. Thus, in the Cypress EEPROM, the "DUT EEPROM file" field was filled in with "differentialsensorboard".

When this board is plugged in, the DLL opens eeprom\differentialsensorboard and uses the contents as the DUT EEPROM.

The new architecture for SensorPath uses a capture board and pluggable DUT boards. Since the DUT EEPROMs may change (or may not even exist), the "DUT EEPROM file" field in the Cypress EEPROM should be left blank; a DUT EEPROM image should NOT be forced for this architecture.